1 Forelesning 5

1.1 Selection Sort

- Finn det minste elementet
- Flytt det først (index 0)
- Finn minste elementet blant de $n-1$ siste.
- Flytt det på andre posisjon (index 1)

$\cdots$ til alle elementene er sortert.

Hva er kjøretiden til denne funksjonen?

```java
public <T extends Comparable<? super T>> void sort(List<T> list) {
    List<T> sorted = new ArrayList<>();

    while (!list.isEmpty()) { // n iterasjoner
        T smallest = Collections.min(list); // O (n)
        list.remove(smallest); // O (n)
        sorted.add(smallest); // O (n)
    }
    list.addAll(sorted);
}
```

Figure 1: Selection sort har kjøretid $O\left(n^2\right)$

1.2 ”In-Place” Sortering

- Sortere uten å bruke ekstra minne
- Ikke lag ny liste, men bytt elementer i listen
- Dette forbedrer ikke big-O kjøretid, men kan redusere konstantene i kjøretiden
- Dette kalles optimering av kode

1.3 Insertion Sort

- Lag en ny liste
- Gå gjennom elementene

1.4 Quick Sort

- Finn et element kalt ”pivot”
- Divide:
  - Første list $\leq$ pivot

  - Siste list $\geq$ pivot
  - Hva med de lik pivot?
- Sort hver liste rekursivt.
- Slå sammen disse listene
  - Siden vi delte inn i basert på pivot blir det
  - Første, pivot, siste
- Hva er kjøretiden til QuickSort
  - Det ligner litt på merge sort
  - Vi deler opp i to lister, men er de like store?
  - Hvor mange ganger må vi dele opp?
- I verste tilfelle blir pivot det største eller minste elementet?
- I beste tilfelle blir pivot det midterste elementet.

Quicksort har altså runtime $T \in (O(n \log(n)), O(n^2))$